Wirtualizacja rozproszonej pamięci operacyjnej multikomputera dla systemu Linux w oparciu o koncepcję SDDS

Autoreferat Rozprawy Doktorskiej

Autor: mgr inż. Arkadiusz Chrobot

Promotor: prof. dr hab. inż. Krzysztof Sapiecha

1. Wprowadzenie

Multikomputery (wielokomputery) są systemami wieloprocesorowymi, zbudowanymi z pojedynczych komputerów (węzłów), które nie mają wspólnej przestrzeni adresowej, a wymiana informacji między nimi dokonywana jest wyłącznie za pomocą przesyłania komunikatów w lokalnej sieci komputerowej [1]. W takich systemach ważnym zagadnieniem jest efektywne zarządzanie rozproszonymi zasobami, w szczególności rozproszoną pamięcią operacyjną. Wirtualizacja, rozumiana jako agregacja zasobów, jest techniką, która umożliwia zrealizowanie tego zadania. W odniesieniu do rozproszonej RAM ma ona trzy dziedziny zastosowań: wsparcie dla pamięci operacyjnej, rozproszoną pamięć dzieloną i wsparcie dla usług zdalnych [2].

Wsparcie dla pamięci operacyjnej ma na celu wykorzystanie agregacji wolnej RAM (ang. *idle* RAM) poszczególnych węzłów multikomputera do stworzenia wspólnego zasobu z którego mogą one korzystać według potrzeb. Taka pula pamięci jest wykorzystywana najczęściej jako przestrzeń wymiany dla klasycznych implementacji pamięci wirtualnej lub jako urządzenie blokowe. Przykładami rozwiązań należących do tego obszaru zastosowania wirtualizacji rozproszonej RAM są Remote Memory Model [3], Distributed Anemone [4, 5], Nswap [6] i The Network RamDisk [7].

Rozproszona pamięć dzielona (ang. *Distributed Shared Memory* - DSM) jest rozwiązaniem pozwalającym na zastosowanie mechanizmu komunikacji międzyprocesowej opartego o pamięć dzieloną w multikomputerach, które są takiej pamięci pozbawione. Wyróżnia się dwa poziomy systemu komputerowego, w ramach których może zostać zrealizowany mechanizm rozproszonej pamięci. Pierwszym poziomem jest sprzęt. Na tym poziomie zaimplementowane są takie rozwiązania, jak DASH [8] czy Scalable Coherent Interface (SCI) [9–11]. Kolejny poziom, na którym może zostać zrealizowana DSM, to oprogramowanie. Przykładami takiej realizacji są IVY [12] i TreadMarks [13]. Istnieją także hybrydowe implementacje DSM, tzn. zrealizowane zarówno na poziomie sprzętu jak i oprogramowania. Należą do nich między innymi PLUS [14] i FLASH [15].

Wirtualizacja rozproszonej RAM multikomputera może zostać zastosowana również do wspierania usług zdalnych, takich jak transakcyjne bazy danych, serwery aplikacji lub treści multimedialnej. Oracle Coherence Data Grid [16, 17] jest pośrednią warstwą oprogramowania, która tworzy z rozproszonej RAM pamięć podręczną, która jest w stanie pomieścić zbiory robocze aplikacji świadczące usługi zdalne. Podobną funkcjonalnością dysponują produkty firmy RNA Networks [18]. Skalowalne, Rozproszone Struktury Danych (ang. *Scalable Distributed Data Structures* - SDDS) [19–21], które również należą do tej kategorii rozwiązań, dostarczają bardziej podstawowej funkcjonalności - umożliwiają wydajne i skalowalne przechowywanie rekordów danych w dostępnej, rozproszonej pamięci multikomputera.

Wymienione rozwiązania realizują różne cele, zależnie od grupy, do której należą. Można jednak dokonać ich porównania uwzględniając takie cechy jak: skalowalność, wydajność i odporność na błędy (poprawność). Takie porównanie zostało dokonane w podsumowaniu Rozdziału 2 rozprawy. Na jego podstawie ustalono, że SDDS lepiej wypełniają wymagania stawiane przez multikomputery niż pozostałe z opisanych rozwiązań.

2. Skalowalne, Rozproszone Struktury Danych

SDDS są metodą wirtualizacji rozproszonej pamięci operacyjnej multikomputera na poziomie aplikacji użytkownika [19]. Część węzłów pełni rolę klientów SDDS, a pozostałe serwerów SDDS. Podstawową jednostką danych stosowaną w SDDS jest rekord. Rekordy są grupowane w większe struktury nazywane wiaderkami (ang. *bucket*), a te z kolei tworzą plik SDDS. Zwykle serwer SDDS nadzoruje pojedyncze wiaderko, które rezyduje w jego pamięci operacyjnej. Jeśli na skutek wstawiania nowych rekordów pojemność wiaderka zostanie przekroczona, to następuje jego podział, w wyniku którego powstaje nowe wiaderko, które przejmuje około połowę rekordów od przepełnionego wiaderka. W ten sposób rozmiar pliku SDDS rośnie. Każdy klient SDDS zna pewne parametry, które nazywane są obrazem pliku SDDS. Ich wartości używane są do wyznaczania adresu serwera, który jest w posiadaniu wiaderka zawierającego określony rekord. Skalowalne, Rozproszone Struktury Danych muszą spełniać trzy podstawowe założenia [20]:

1. Do adresowania danych nie może być używany żaden centralny katalog.
2. Obraz pliku SDDS jaki posiada klient może ulec przedawnieniu. Jego aktualizacja dokonywana jest za pomocą komunikatów korygujących nazywanych IAM (ang. *Image Adjustment Message*).
3. Klient z nieaktualnym obrazem może wysłać zapytanie do niewłaściwego serwera SDDS, który musi w takim wypadku przesłać klientowi komunikat IAM i skierować jego żądanie do właściwego adresata.

SDDS posiadają szereg zalet, które są ważne z punktu widzenia systemów multikomputerowych. Zostały one wymienione w Rozdziale 2 rozprawy. Podstawową wadą Rozproszonych, Skalowalnych Struktur Danych jest to, że ich zastosowanie wymaga pracochłonnych modyfikacji istniejących aplikacji użytkowych tak, aby dostosować je do współpracy z tymi strukturami.

3. Teza i cel pracy

Między SDDS, a klasyczną realizacją pamięci wirtualnej, jaką jest stronicowanie na żądanie można zauważyć pewne podobieństwa (Tabela 1), ale Skalowalne, Rozproszone Struktury Danych są rozwiązaniem implementowanym na poziomie aplikacji użytkowych, a pamięć wirtualna jest realizowana na poziomie systemu operacyjnego i sprzętu. To czyni ją przezroczystą (w przypadku stronicowania na żądanie) dla aplikacji użytkownika, podczas gdy użycie SDDS wymusza wprowadzenia do nich modyfikacji. Teza rozprawy zakłada, że można uniknąć tej niedogodności przenosząc zarządzanie wiaderkami i rekordami w SDDS na poziom systemu operacyjnego. W celu udowodnienia tezy opracowano prototypową implementację takiej wersji SDDS. Pracę nad nią podzielono na następujące etapy:

1. analizę możliwości dokonania przeniesienia SDDS na poziom systemu operacyjnego,
2. zaprojektowanie architektury rozwiązania,

Stronicowanie na żądanie	**SDDS**
Rozmiar strony jest stały.	Rozmiar rekordu może być stały.
W przypadku braku miejsca w pamięci operacyjnej część stron jest wycofywana do pamięci pomocniczej (urządzenia wymiany). Operacją tą steruje algorytm wymiany.	W przypadku braku miejsca w wiaderku część rekordów przesyłana jest do innego wiaderka. Operacją tą steruje algorytm podziału wiaderka.
Adresy wirtualne stron przekształcane są na adresy fizyczne. Translacji dokonuje jednostka MMU.	Klucz rekordu zamieniany jest na adres logiczny wiaderka. Translacji dokonuje klient SDDS.

Tabela 1: Analogie między stronicowaniem na żądanie, a SDDS

3. wykonanie prototypowej implementacji,

4. dokonanie eksperymentalnej oceny prototypu.

4. Realizacja celu pracy

W wyniku analizy problemu (Rozdziały 3 i 4 rozprawy) ustalono, że:

- aby użytkowanie SDDS było przezroczyste dla aplikacji użytkownika, to funkcjonalność takich struktur musi być udostępniona tym aplikacjom za pomocą znanego im API,
- nie wszystkie elementy SDDS muszą zostać przeniesione na poziom systemu operacyjnego,
- docelowym systemem dla którego zostanie opracowany prototyp rozwiązania będzie Linux, ze względu na dostępność jego kodu źródłowego [22] oraz szerokie zastosowanie w systemach multikomputerowych [23],
- wyjściową architekturą na podstawie które zostanie opracowane rozwiązanie będzie SDDS LH*.

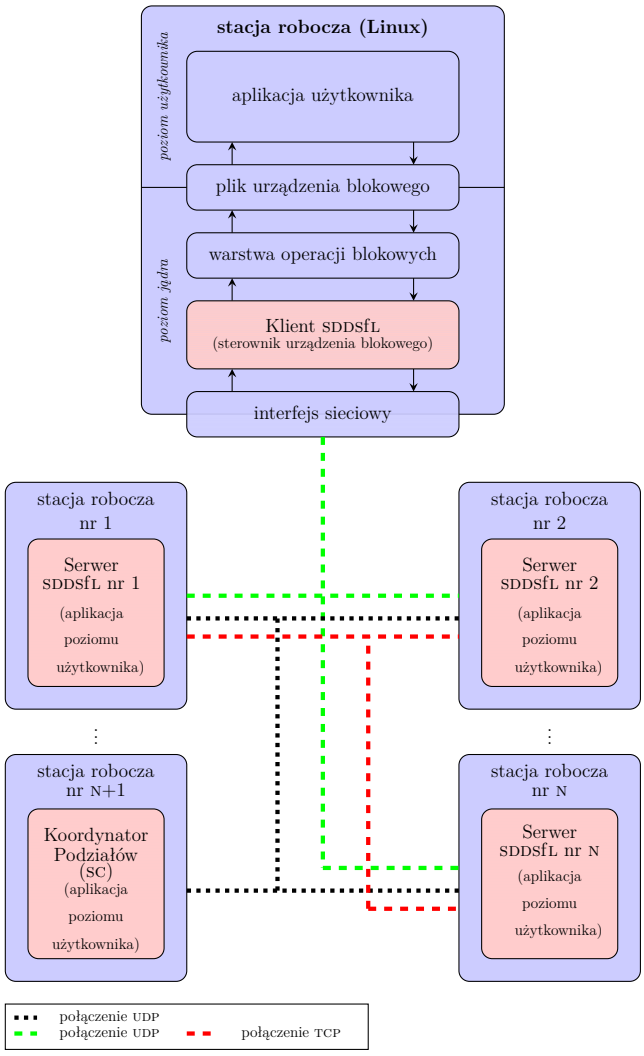
Wymóg użycia znanego aplikacjom interfejsu ogranicza możliwości realizacji udostępnienia funkcjonalności SDDS do trzech podsystemów jądra Linuksa: wywołań systemowych (ang. *system calls*), modułów systemów plików i sterowników urządzeń blokowych.

Udostępnienie usług SDDS za pomocą pierwszego z wymienionych podsystemów wymagałoby dodatnia nowych wywołań do zbioru już istniejących lub dokonania zmian w funkcjach, które je implementują. Każde z tych rozwiązań pociąga za sobą konieczność dokonania znaczących zmian w kodzie jądra systemu, a w przypadku dodania nowych wywołań również modyfikacji aplikacji użytkowych.

Można jednak użyć podzbioru istniejących wywołań systemowych bez dokonywania tak dużych zmian. Tę możliwość oferują moduły obsługi systemów plików oraz sterowniki urządzeń blokowych [24]. Spośród tych opcji bardziej elastyczną w zastosowaniu jest sterownik urządzenia blokowego - tworzy on wirtualne urządzenie blokowe, na którym można

osadzić dowolny, lokalny system plików, zależnie od potrzeb aplikacji użytkowych. Dodatkowo może ono pracować jako urządzenie wymiany (ang. *swap device*) dla stronicowania na żądanie.

Architekturę rozwiązania spełniającego wymogi tezy rozprawy, nazwanego SDDSfL, oparto o Skalowalną, Rozproszoną Strukturę Danych LH* (SDDS LH*) [25]. Jest to wariant SDDS w którym do adresowania wiaderek użyto algorytmu haszowania liniowego [26] uogólnionego na środowiska rozproszone. Algorytm ten nosi nazwę LH*. W SDDS LH* oprócz serwerów i klientów występuje dodatkowy element. Jest to koordynator podziałów (ang. *Split Coordinator* - SC), który wyznacza kolejność podziału wiaderek. Na etapie projektowania architektury SDDSfL ustalono, że jedynym jej elementem, który zostanie umieszczony na poziomie jądra systemu operacyjnego będzie klient. Zarówno serwery, jak i koordynator podziałów pozostają oprogramowaniem działającym w przestrzeni użytkownika. Rysunek 1 przedstawia ogólny schemat SDDSfL. Architektura SDDSfL została opisana w Rozdziale 4 rozprawy.



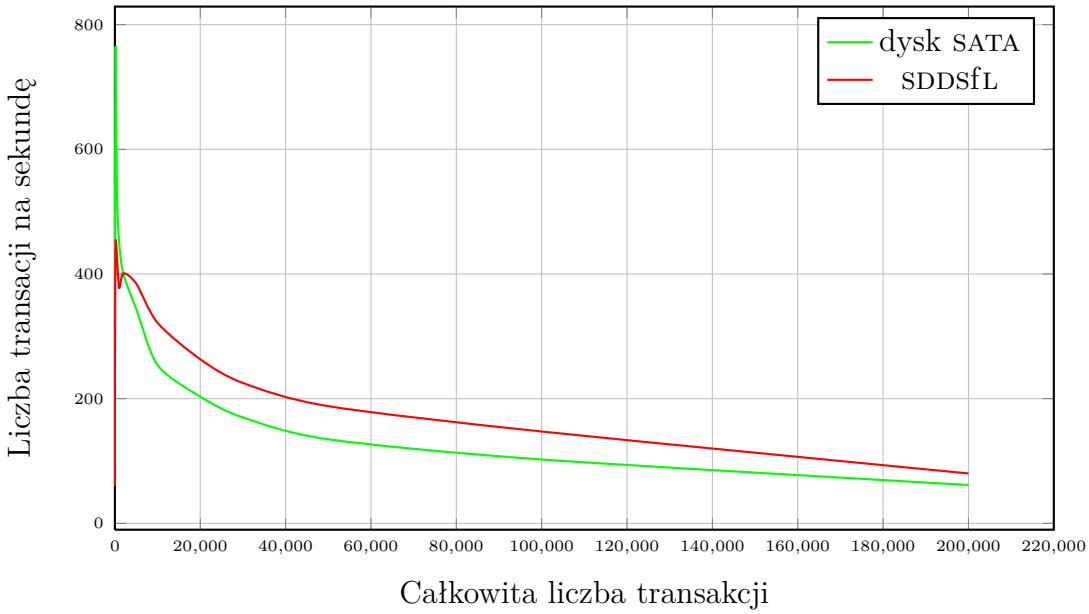
Rysunek 1: Architektura SDDSfL

W prototypowej implementacji SDDSfL klient został zrealizowany w postaci modułu jądra systemu operacyjnego, który zgodnie z przyjętą architekturą, jest sterownikiem

urządzenia blokowego. Serwery i koordynator podziałów są aplikacjami użytkowymi. Rolę rekordów w SDDSfL pełnią bloki danych. Do zarządzania nimi wewnątrz wiaderek użyte zostało adresowanie otwarte z podwójnym haszowaniem[1] [27, 28]. Opis implementacji SDDSfL został zamieszczony w Rozdziale 5 rozprawy.

5. Ocena eksperymentalna

SDDSfL jest widziane przez aplikacje użytkowe jako urządzenie blokowe, zatem porównano je z rozwiązaniami postrzeganymi przez oprogramowanie użytkowe w podobny sposób, czyli z dyskami twardymi i rozproszonymi systemami plików. Testy przeprowadzono dla aplikacji zorientowanych na wejście-wyjście i wymagających swobodnego dostępu do danych (transakcyjne bazy danych, sortowanie plików rekordów przy pomocy algorytmu QuickSort) oraz dla procesów zorientowanych na obliczenia i korzystających z sekwencyjnego dostępu do danych (wyszukiwanie wzorca w pliku tekstowym). W testach użyto dwóch sieci lokalnych - InfiniBand [29] i Gigabit Ethernet. Na wykresach, jeśli to było konieczne, oznaczono je jako (inf) i (gbe). Maksymalna szybkość przesyłania danych dla zastosowanej sieci InfiniBand wynosi 1,25 GiB/s. Testowano również użycie SDDSfL jako urządzenia wymiany dla stronicowania na żądanie. Wyniki testowania wydajności transakcyjnej bazy danych osadzonej na SDDSfL i na dysku SATA przedstawione są na wykresie 2. W teście użyto bazy danych PostgreSQL [30] i oprogramowania `pgbench` [31]. Z wy-

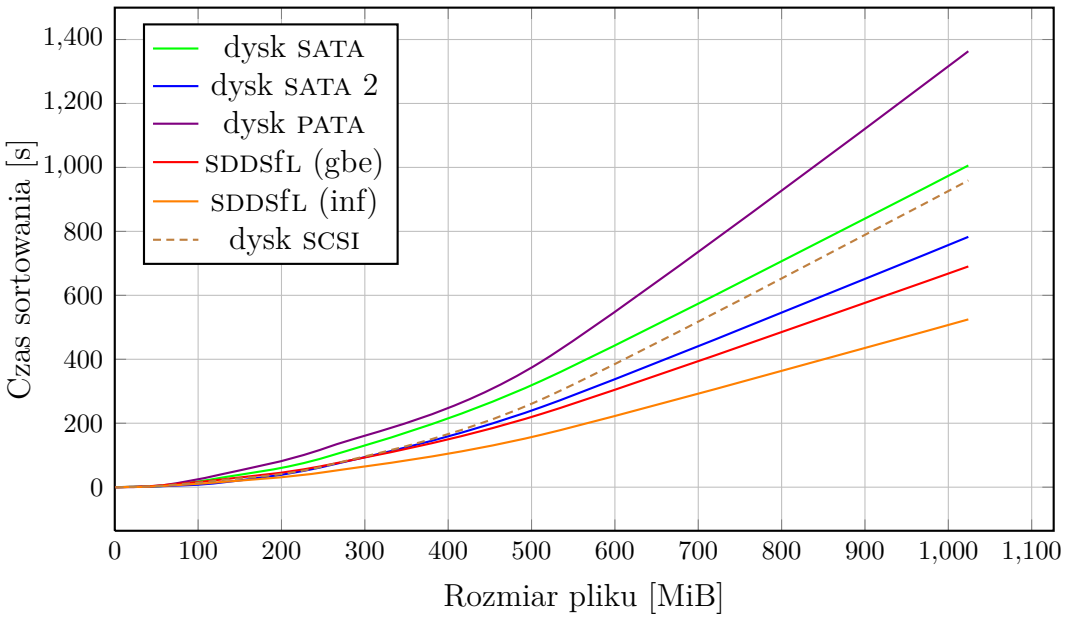


Rysunek 2: Wydajność transakcji dla bazy danych PostgreSQL

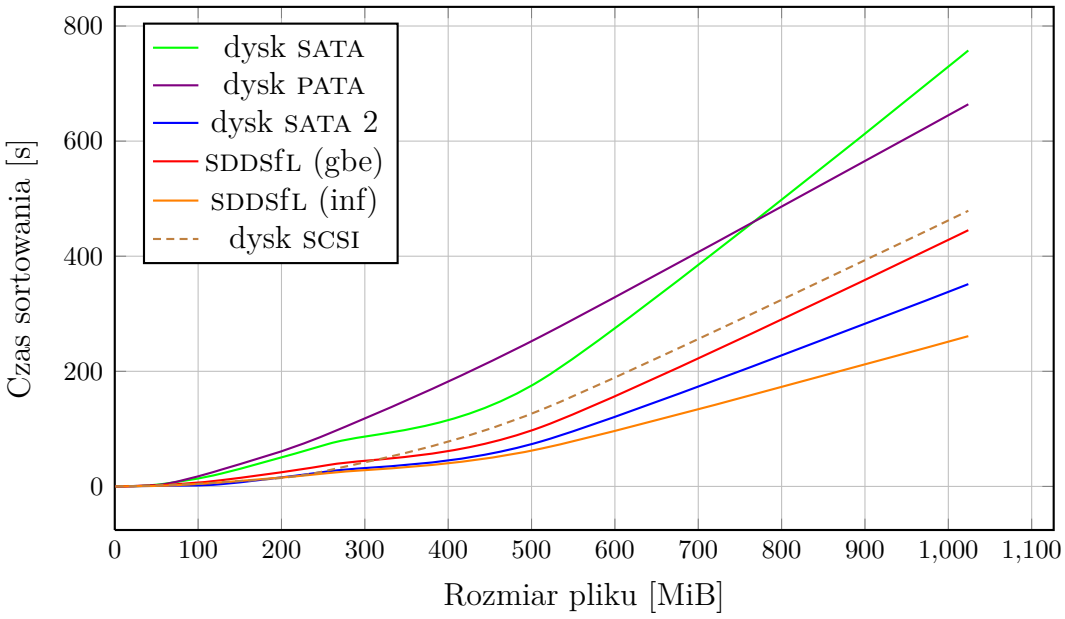
kresu wynika, że baza danych osiąga większą liczbę transakcji na sekundę, jeśli korzysta z SDDSfL.

Wykresy 3 i 4 zawierają wyniki pomiaru czasu sortowania plików z użyciem SDDSfL i zestawu dysków twardych.

[1] Architektura SDDS LH* określa sposób wyznaczania adresów wiaderek, ale algorytm zarządzania danymi wewnątrz wiaderka może być wybrany dowolnie.

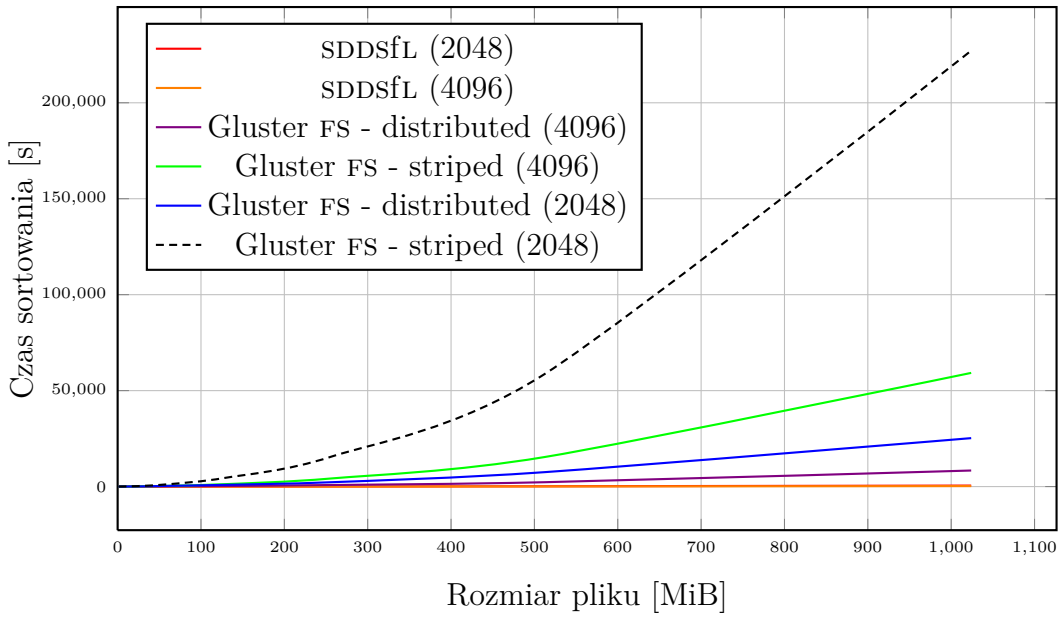


Rysunek 3: Czas sortowania plików o rozmiarze rekordów 2 KiB



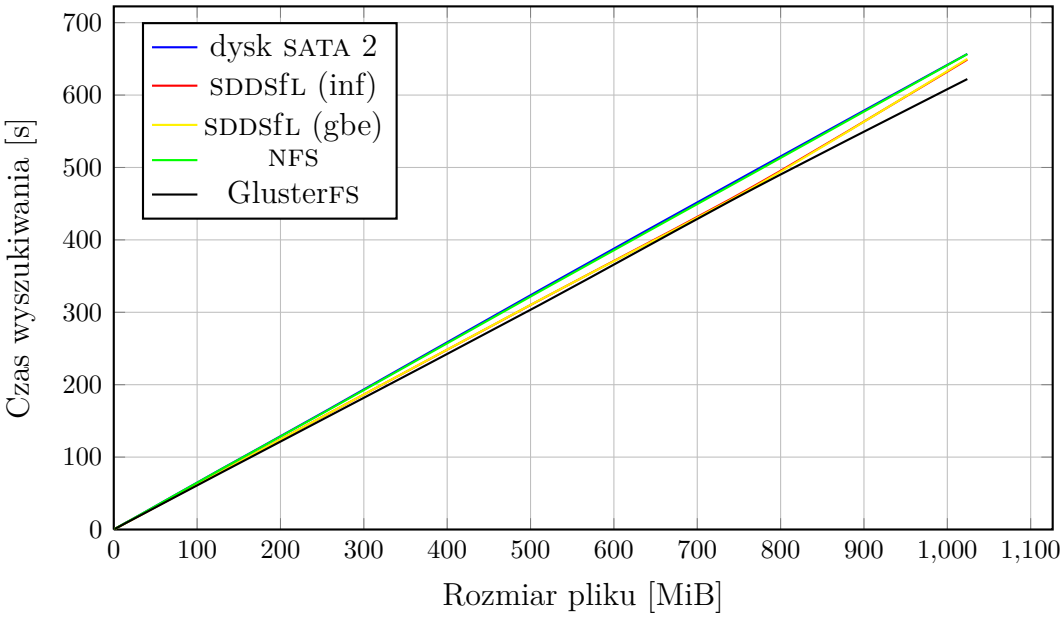
Rysunek 4: Czas sortowania plików o rozmiarze rekordów 4 KiB

Analiza rezultatów wskazuje na dobrą wydajność SDDSfL w porównaniu z dyskami twardymi. Jedynie nowsze dyski, z łączem SATA 2 uzyskiwały w niektórych testach porównywalną wydajność. Podobne testy przeprowadzono dla rozproszonego systemu plików GlusterFS [32]. W tych testach SDDSfL osiągnęło znaczącą przewagę (Wykres 5).

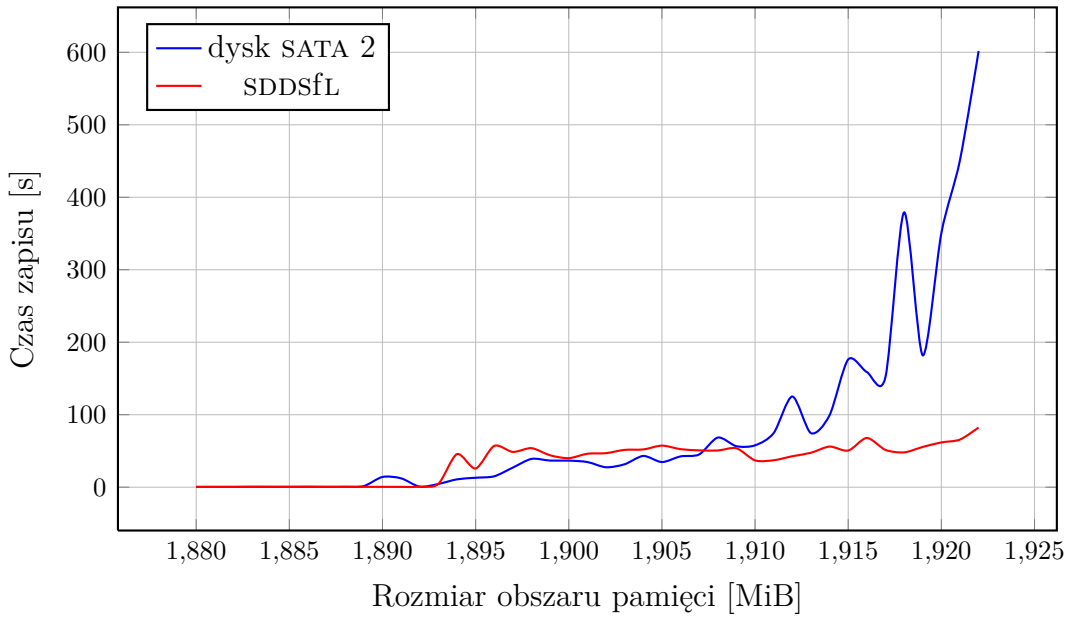


Rysunek 5: Czas sortowania plików dla SDDSfL i GlusterFS (w legendzie wykresu podano w nawiasach rozmiary rekordów w sortowanych plikach)

Testy z użyciem aplikacji wymagających sekwencyjnego dostępu do danych nie wykazały znaczącej przewagi żadnego z testowanych rozwiązań (Wykres 6).



Rysunek 6: Wyszukiwanie wzorca w pliku tekstowym



Rysunek 7: Czas zapisu dużego obszaru RAM

Porównano także wydajność SDDSfL pracującego w charakterze urządzenia wymiany z wydajnością dysku twardego użytego w tej samej roli. Wyniki przedstawione są na Wykresie 7. Można z nich wywnioskować, że czas wymiany strony jest krótszy jeśli przestrzeń wymiany jest zorganizowana na urządzeniu SDDSfL. Wynik ten jest porównywalny z opublikowanymi w artykułach [4–6] rezultatami takich rozwiązań, jak Distributed Anemone, MemX i Nswap.

Oprócz testów wydajnościowych wykonano także testy oceniające skalowalność SDDSfL. Zgodnie z oczekiwaniami potwierdziły one wysoki stopień skalowalności tego rozwiązania.

Szczegółowa analiza wyników i rezultaty pozostałych testów zostały zamieszczone w Rozdziale 6 rozprawy.

6. Podsumowanie

Główne oryginalne osiągnięcia autora rozprawy to:

1. Opracowanie rozwiązania (SDDSfL), które umożliwia korzystanie aplikacjom użytkowym w sposób przezroczysty z funkcjonalności SDDS. Podstawowym obszarem zastosowania SDDS jest wsparcie dla usług zdalnych. To rozwiązanie rozszerza zakres zastosowań SDDS na obszar wspomagania dla pamięci operacyjnej.

2. Zaprojektowanie architektury Skalowalnych, Rozproszonych Struktur Danych dla Linuksa, która może być uogólniona na inne systemy operacyjne, które wspierają paradygmat urządzeń blokowych.

3. Wykonanie prototypowej implementacji SDDSfL, która jest wydajna i skalowalna.

Literatura (wybrane pozycje z bibliografii rozprawy)

[1] Andrew Stanley Tanenbaum. *Systemy operacyjne*. Helion, Gliwice, 2010.

[2] Clive Cook. *Memory Virtualization, the Third Wave of Virtualization*. http://vmblog.com/archive/2008/12/14/memory-virtualization-the-third-wave-of-virtualization.aspx, 2009.

[3] Douglas E. Comer, J Griffioen. A New Design for Distributed Systems: The Remote Memory Model. *Proceedings of the Summer 1990 Usenix Conference*, 1990.

[4] Michael R. Hines, Jian Wang, Kartik Gopalan. Distributed Anemone: Transparent Low-Latency Access to Remote Memory. *HiPC*, strony 509–521, 2006.

[5] Michael R. Hines, Kartik Gopalan. MemX: supporting large memory workloads in Xen virtual machines. *VTDC '07: Proceedings of the 3rd international workshop on Virtualization technology in distributed computing*, strony 1–8, New York, NY, USA, 2007.

[6] Tia Newhall, Sean Finney, Kuzman Ganchev, Michael Spiegel. Nswap: A network swap module for linux clusters. *Proceedings of the International Conference on Parallel and Distributed Computing*, wolumen 2790, strony 1160–1169, 2003.

[7] Michail D. Flouris, Evangelos P. Markatos. The Network RamDisk : Using Remote Memory on Heterogeneous NOWs. *Cluster Computing*, 2:281–293, 1999.

[8] Daniel Lenoski, James Laudon, Kourosh Gharachorloo, Wolf-dietrich Weber, Anoop Gupta, John Hennessy, Mark Horowitz, Monica S. Lam. The Stanford DASH multiprocessor. *IEEE Computer*, 25:63–79, 1992.

[9] D.B. Gustavson, Qiang Li. The Scalable Coherent Interface (SCI). *Communications Magazine, IEEE*, 34(8):52–63, 1996.

[10] David V. James, Anthony T. Laundrie, Stein Gjessing, Gurindar Sohi. Scalable coherent interface. *Computer*, 23(6):74–77, 1990.

[11] Oliver Pugh, Jeff Cowhig, Jonathan Crowe, Eoin Blacklock. *Scalable Coherent Interface (SCI)*. http://ntrg.cs.tcd.ie/undergrad/4ba2.05/group12/index.html, 2010.

[12] Kai Li, Paul Hudak. Memory coherence in shared virtual memory systems. *ACM Trans. Comput. Syst.*, 7(4):321–359, 1989.

[13] Pete Keleher, Alan L. Cox, Sandhya Dwarkadas, Hya Dwarkadas, Willy Zwaenepoel. TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems. *In Proceedings of the 1994 Winter Usenix Conference*, strony 115–131, 1994.

[14] Roberto Bisiani, Mosur Ravishankar. PLUS: a distributed shared-memory system. *SIGARCH Comput. Archit. News*, 18(3a):115–124, 1990.

[15] Mark Heinrich, Jeffrey Kuskin, David Ofelt, John Heinlein, Joel Baxter, Jaswinder Pal Singh, Richard Simoni, Kourosh Gharachorloo, David Nakahira, Mark Horowitz, Anoop Gupta, Mendel Rosenblum, John Hennessy. The performance impact of flexibility in the Stanford FLASH multiprocessor. *ASPLOS-VI: Proceedings of the sixth international conference on Architectural support for programming languages and operating systems*, strony 274–285, New York, NY, USA, 1994. ACM.

[16] Cameron Purdy. *Getting Coherence: Introduction to Oracle Coherence Data Grid*. http://www.youtube.com/watch?v=4Sq45B8wAXc, 2007.

[17] Oracle Coherence Knowledge Base Home. http://coherence.oracle.com, 2009.

[18] Tom Matson. *RNA Networks - Virtual Memory*. http://www.motionbox.com/videos/0096d1b61d12efc58f, 2009.

[19] Witold Litwin, Marie-Anna Neimat, Donovan A. Schneider. LH* — a scalable, distributed data structure. *ACM Transactions on Database Systems*, 21(4):480–525, 1996.

[20] W. Litwin, M-A Neimat, D. Schneider. RP*: A Family of Order Preserving Scalable Distributed Data Structures. *Proceedings of the Twentieth International Conference on Very Large Databases*, strony 342–353, Santiago, Chile, 1994.

[21] Cédric du Mouza, Witold Litwin, Philippe Rigaux. SD-Rtree: A scalable distributed rtree. *ICDE*, strony 296–305, 2007.

[22] Linux Kernel. http://kernel.org/, 2010.

[23] TOP500 Supercomputing Sites. http://www.top500.org, 2010.

[24] Robert Love. *Kernel Linux - Przewodnik programisty*. Wydawnictwo Helion, Gliwice, 2004.

[25] Arkadiusz Chrobot, Grzegorz Łukawski, Krzysztof Sapiecha. Scalable Distributed Data Structures for Linux-based Multicomputer. *International Symposium on Parallel and Distributed Computing in Kraków, IEEE Computer Society*, strony 424–428, 2008.

[26] Witold Litwin. Linear hashing: a new tool for file and table addressing. *VLDB '1980: Proceedings of the sixth international conference on Very Large Data Bases*, strony 212–223. VLDB Endowment, 1980.

[27] Donald E. Knuth. *Sortowanie i wyszukiwanie*, wolumen 3 serii *Sztuka programowania*. WNT, Warszawa, 2002.

[28] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest. *Wprowadzenie do algorytmów*. Wydawnictwa Naukowo-Techniczne, Warszawa, 1998.

[29] InfiniBand Trade Association. http://www.infinibandta.org, 2010.

[30] PostgreSQL. http://postgresql.org, 2010.

[31] pgbench (Postgres Benchmark). http://wiki.postgresql.org/wiki/Pgbench, 2010.

[32] GlusterFS. http://www.gluster.org, 2010.